Bootstrapping a Trustworthy and Seamless Digital Engineering System

Ph.D. Preliminary Exam

James S. Wheaton

Advised by: Dr. Daniel Herber

Department of Systems Engineering
Walter Scott, Jr. College of Engineering
Colorado State University

April 17, 2023

Committee Members:
Dr. Erika Miller
Dr. Steve Simske
Dr. Vinayak Prabhu

Outline

→ James S. Wheaton

- B.S. Mechanical Engineering, Purdue University (2011)
- Former software engineer and consultant in ecommerce, big data, and blockchain
- Started Systems Engineering Ph.D. @ CSU in 2017, part-time remote
- Completing coursework Spring 2023 in the 72-credit-hour Ph.D. degree program
- Computer hobbyist since age 5
- Likes to study programming languages of all kinds
- Builds all software from source with hardened toolchains, whereever possible

① Introduction

→ Digital Engineering Is an Integration Challenge

Digital Engineering currently relies on *interoperability* as the primary mechanism for constructing the Authoritative Source of Truth, e.g. with APIs and format interchange standards[1].

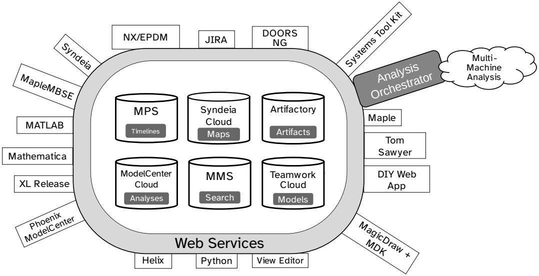


Figure: Depiction of NASA JPL OpenCAE Environment (Adapted from Delp 2019)

[1] Bajaj, Friedenthal, and Seidewitz 2022

→ We Build Our Computer Systems Like Cities

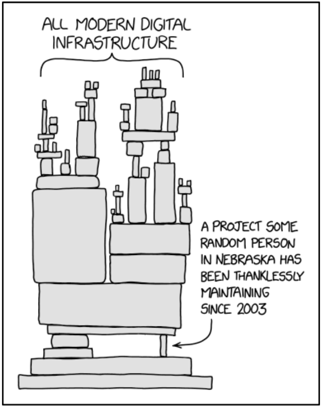


Figure: xkcd: Dependency (Munroe 2020)

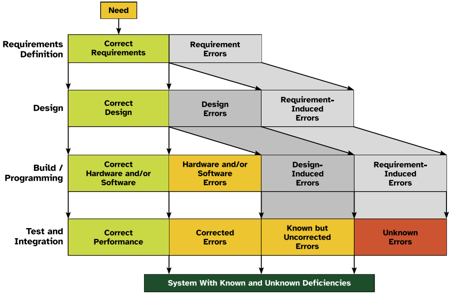


Figure: The Error Avalanche (Adapted from Claxton, Cavoli, and C. Johnson 2005)

→ Cybersecurity is a “Mess” or Wicked Problem

US CISA Director has recently highlighted a *“normalization of deviance”* in the computing industry and called on vendors to provide systems that are **secure-by-default** and **secure-by-design**.[1] The situation is untenable:

- Pervasive use of memory-unsafe languages, including in OS, compilers, security-critical components and theorem provers[2]
- Common Vulnerabilities and Exposures are on the rise
- CPU microarchitecture side-channel vulnerabilities are unpatchable[3]
- Internet architecture vulnerabilities & protocol ossification[4]
- Trusting Trust attack remains ignored after 50 years[5]
- Cyber-infrastructure is inherently insecure [6]

[1] Easterly 2023 [2] Chisnall 2018; Du, Wu, and Mao 2023; Winterer, Zhang, and Su 2020; Bringolf, Winterer, and Su 2022 [3] Porras and Lindell 1995; Lipp et al. 2020; Kocher et al. 2020; Schwarz, Weiser, Gruss, et al. 2017; Van Bulck et al. 2018; Weisse et al. 2018; Schwarz, Weiser, and Gruss 2019; Skarlatos et al. 2019; Murdock et al. 2020; Schaik, Minkin, et al. 2021; Schaik, Kwong, et al. 2020; Borrello et al. 2022 [4] Ammar 2018; Papastergiou et al. 2016 [5] Karger and Schell 2002; Thompson 1984; Wheeler 2009 [6] Massacci, Jaeger, and Peisert 2021; Smith and Mulrain 2018; Dawson et al. 2021; Algarni 2021; Hobbs 2021

Digital Engineering Has a Reverse Salient

The *reverse salient* is a **set of critical problems**[1] whereby system components "fail to deliver the necessary level of technological performance thereby inhibiting the performance delivery of the system as a whole":[2]

- WIMP applications paradigm — essential functions are outsourced
- false dichotomy of user / developer
- inscrutable binary executable vs. sprawling source code
- physical centralization + lack of isolation, e.g. CPU
- sequential-first processing, e.g. CPU
- lack of integrated program documentation, test, and verification facilities
- plethora of ill-defined languages/formats
- security-by-obscurity

[1] Hughes 1993 [2] Dedehayir and Mäkineif 2008

→ Looking for the Escape Hatch

The systemic problem of trustworthy cyber-systems has been known for 25 years.[1]
Recent research efforts have attacked the mess from different perspectives:

- Fully Countering Trusting Trust through Diverse Double-Compiling[2]
- DARPA Cyber-Assured Systems Engineering (CASE)[3]
- DARPA Clean-slate design of Resilient, Adapative, Secure Hosts (CRASH)[4]
- DARPA META-II[5]
- DARPA Circuit Realization At Faster Timescales (CRAFT)[6]
- Deep Specification[7]
- Formally-verified stack from assembly language to CPU[8]

[1] McLean 1997; Council et al. 1999; Mundie et al. 2002; Spafford 2004 [2] Wheeler 2009 [3] Cofer 2021 [4] *Clean-slate design of Resilient, Adapative, Secure Hosts (CRASH)* 2010; Chiricescu et al. 2013 [5] *META-II* 2010 [6] *Circuit Realization At Faster Timescales (CRAFT)* 2015 [7] Appel et al. 2017 [8] Moore 2003; Moore 2007

→ Research Questions

RQ1 What are the gaps, barriers and cost drivers of engineering provably-correct cyber-systems?

RQ2 Can these gaps be adequately addressed with today's computing ecosystem?

RQ3 What would a clean-slate digital engineering system that addresses the gaps and barriers look like?

RQ4 Can we prove that such an architecture is seamless and trustworthy?

②

Operational Concept of Clean-slate DE System

→ DE Meta-Model

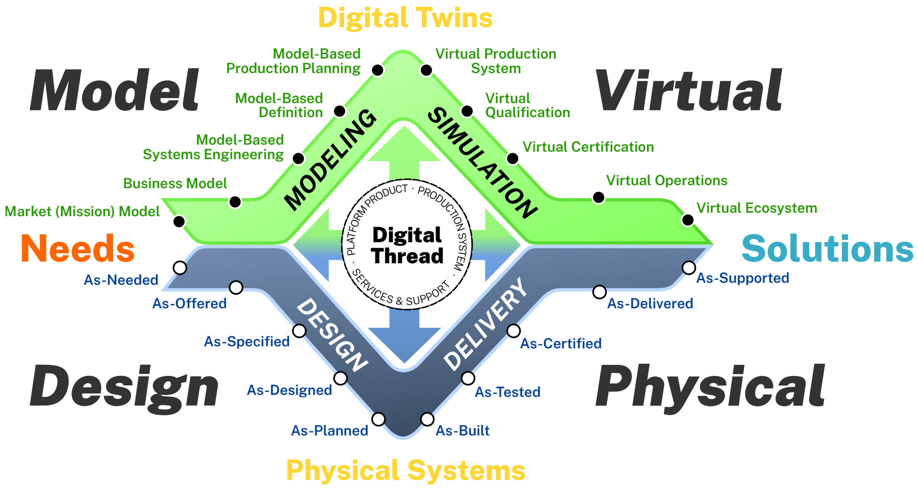


Figure: The Boeing MBSE Diamond: Continuity of the system's 'Digital Thread' (Adapted from Seal 2018)

→ DE Essential Functionality

DE practitioners need a predictable set of affordances for doing their work:

- *Mathematics:* matrices, equation solving, calculus, optimization, probability and statistics, discrete math, theorem proving
- *Science:* physical constants & models, simulations, experimental design, properties of matter
- *Engineering:* 3D geometry, finite-element analysis, fluid dynamics, thermodynamics, materials, reliability, systems modeling, units
- *Knowledge Engineering:* ontologies, authoritative data, rich media, process meta-models, query capabilities
- *Project & Program Management:* PERT, critical path method, EVM, Gantt charts, project economics and accounting

→ Human-Computer Interaction

We need to re-think HCI for human factors:

- WIMP breaks down at scale
- Applications enforce costly context switches, data incommensurability
- Everything-is-an-object with Capabilities is a simpler formalism
- Coherence of textual & graphical representations aids efficient, diverse uses
- Localization and accessibility must be designed-in from the beginning
- AI augmentation is an option, powerful in some contexts

→ Quality Attributes

- **seamless:** consistent and coherent interfaces throughout
- **trustworthy:** provenance of components is known, auditable and traceable; components reliably implement their specifications and carry proof certificates
- **elegant:** "a system that is robust in application, fully meeting specified and adumbrated intent, is well-structured, and is graceful in operation"[1]
  - efficacy
  - efficiency
  - robustness
  - minimizing unintended consequences
- **convivial:** serve the operator and their community for creative and autonomous use, with the power to develop mastery[2]

[1] M. D. Watson, Mesmer, and P. Farrington 2019; M. Watson, Mesmer, and P. Farrington 2020; Madni 2012; M. D. Watson, Griffin, et al. 2014; M. D. Watson 2017 [2] Voinea 2018

③

Theory Development

→ Understanding Tool Integration

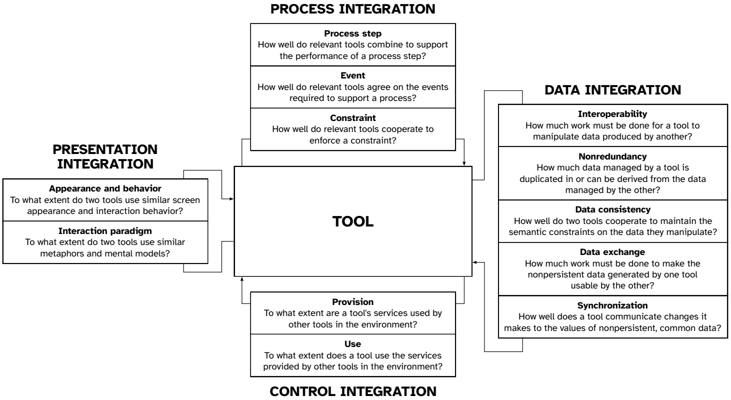


Figure: Tool Integration Entity-Relationship Diagram (Adapted from Thomas and Nejmeh 1992)

→ LISP the Meta-Language

Language-oriented programming has "advantages for domain analysis, rapid prototyping, maintenance, portability, user-enhanceable systems, reuse of development work, while also providing high development productivity" [1]

One of the guidelines of language-oriented programming is that it "enables creators of languages to enforce its variants. ...When a program consists of pieces of different languages, values flow from one context into another and need protection from operations that might violate their integrity."[2]

Programming paradigms depending on need: imperative, functional/declarative, symbolic, constraint/logic, array and stack, dataflow, query, metaprogramming. Gradual typing supports different phases of the system development lifecycle.

[1] Ward 1994 [2] Felleisen et al. 2018

→ Defining Seamless Architecture

- Are Interfaces everywhere fully defined and satisfied at every connection endpoint (Port)?
- Do Parts refine their imported types?
- Do Part Specifications prove out Ports are derivations of internal Parts and `in` Ports and Item Flows?
- Disparate interfaces are not exposed to the operator (“islands of functionality”)

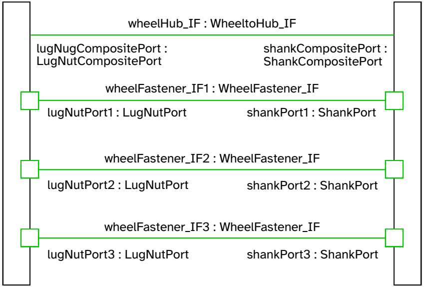


Figure: Interfaces in SysML v2 demonstrating seamlessness (Adapted from Friedenthal 2023)

→ Defining Trustworthy

Trustworthiness is a Quality Attribute related to reliability and security, and based on a set of measurable factors:

- Behavior is well-defined
- Side-channels are explicitly guarded where feasible
- Object Capabilities are ubiquitous for fine-grained security[1]
- Components carry proof certificates, with traceability
- System must be independently verifiable against their specifications
- Bootstrappable, defended against Trusting Trust attacks

[1] Rees 1995; Richardson, Carey, and Schuh 1993

→ Sketching the Bootstrap Process

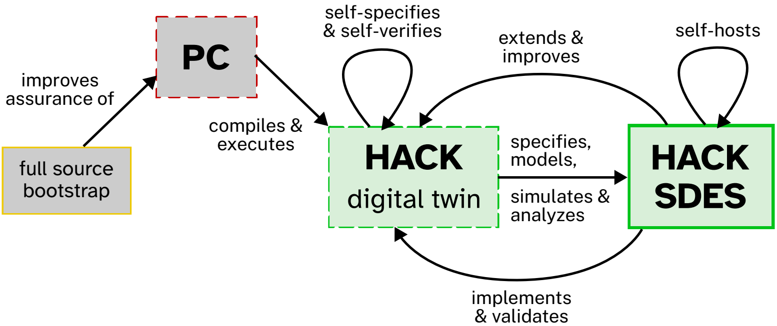


Figure: Simplified view of HACK bootstrap process

→ Formal Proof Strategy

Goal: A trustworthy system is constructible from untrustworthy components.[1]

- Untrustworthy components are diverse
- Untrustworthy components produce the same output for a given input
- Trustworthy components carry proof certificates
- Trustworthy components are auditable
- Untrustworthy components are replaceable by trustworthy components
- Trustworthy system has an independently-verifiable root-of-trust

Build from Wheeler 2009's Diverse Double-Compiling formal proof to include more of the system components.

[1] Rajendran, Sinanoglu, and Karri 2016; Cui et al. 2022; Sethumadhavan et al. 2015

④

Design of High-Assurance Computing Kit

→ HACK Specification Tree

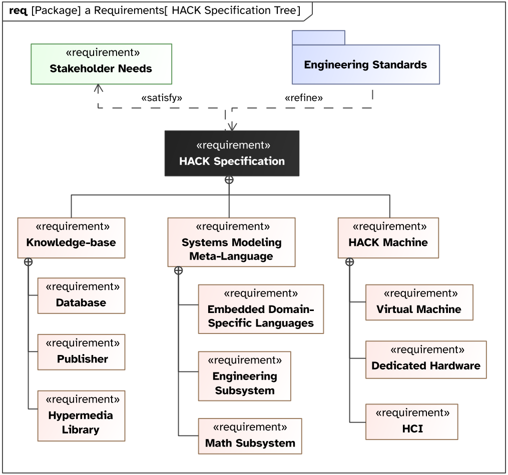
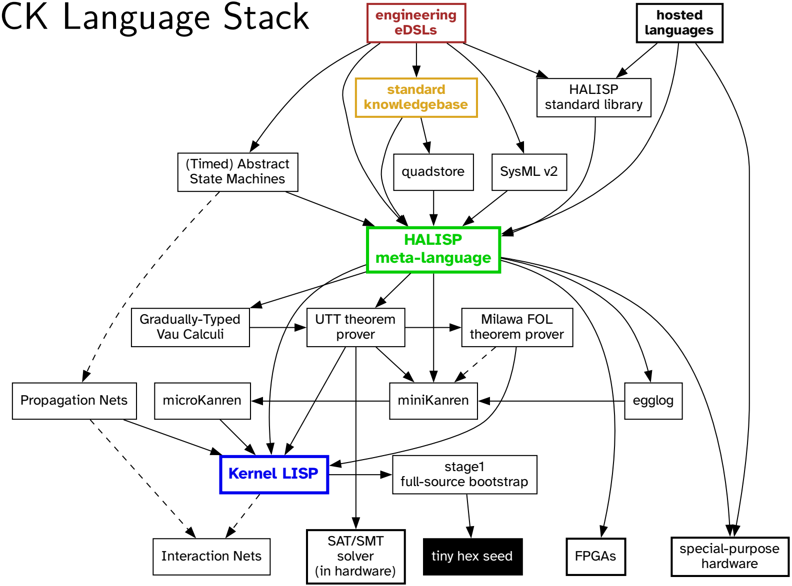


Figure: HACK Specification Tree

→ HACK Language Stack



→ HALISP Language Design

```
(def:function hello-world   ;; name of the pure function
  :doc       (document "A complete DocBook object or AsciiDoc string")
  :type      [ String :-> String ]
  :params    [ name ]
  :requires  [(> (length name) 0)]
  :ensures   (= (length out)  ;; Hoare triple post-condition
                (+ (length "Hello, ")
                   (length name)
                   (length "!")))
  :satisfies [ :FR/001 ]    ;; SysML Block "satisfies" Relationship
  :tests      [(test trivial-example
                 :doc "Test that the name is inserted into the greeting."
                 :verifies [ :FR/009 ]  ;; SysML Functional Requirement
                 (= "Hello, World!"
                    (hello-world "World")))
               (ref:test :T/HW-002)]
  :version   { :major 0 :minor 1 :patch 0 })  ;; enforceable semantic versioning

;; Separation of specification and implementation
(def:function-body hello-world
  (str "Hello, " name "!"))  ;; function body usually starts on a newline
```

Figure: HALISP integrates formal verification, systems eng. & project mgmt.

→ HACK HCI (1)

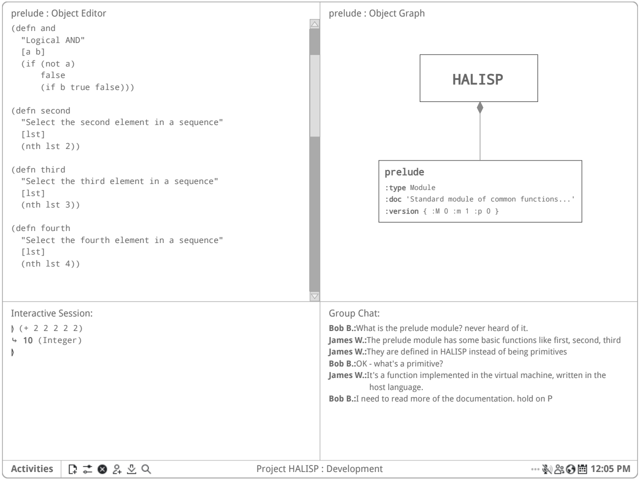


Figure: Team collaboration with text/voice chat and screenshare is built-in

→ HACK HCI (2)

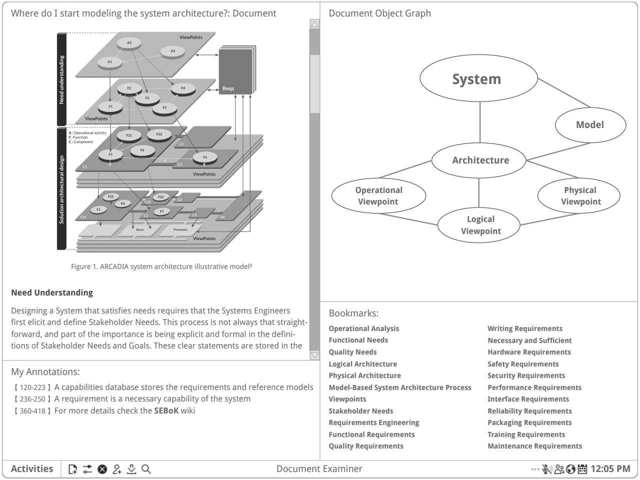


Figure: Knowledgebase is browseable, annotatable, with transclusion & object graph views

→ HACK HCI (3)

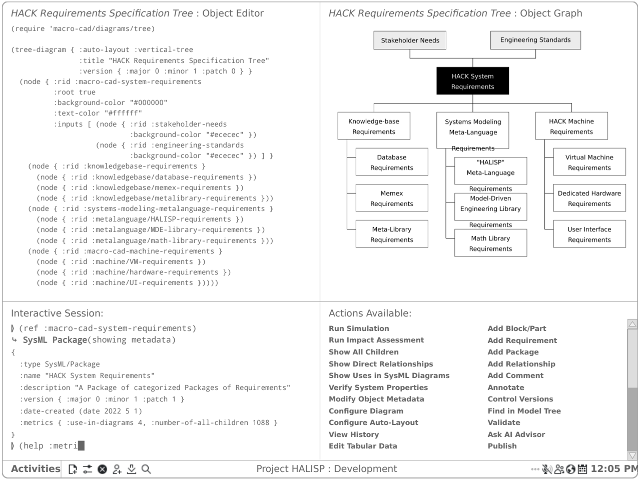


Figure: Textual and graphical representations are coherent; interactive programming session; contextual actions are listed

→ HACK Prototype

The HACK prototype is being developed with the capabilities to model the HACK system. Current functionality is testing the LISP meta-language approach:

- Small LISP implementation with REPL in Ada 2012
- Programmable graphics with SVG for diagrams and presentation slides
- Use of DocBook standard to define systems engineering document templates
- Website generation from given templates and source files
- SysML and project definitions
- Knowledgebase entry definitions

5

Summary & Research Plan

→ A Grand Challenge in Systems Engineering

- Digital Engineering requires a **Transdisciplinary Systems Engineering** approach![1]
- We need the right framing and goal to build support
- End-to-end formal verification enables certification of system and components' correctness so they can be deemed *finished* or safely refactored

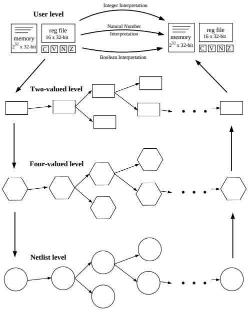


Figure: The FM9001 gate-level model corresponds to the high-level functional model (Moore 2003; Moore 2007)

[1] Mesmer et al. 2022

→ Research Contributions

1. Operational concept of a trustworthy & seamless digital engineering system
2. Formal definitions of ‘trustworthy’ and ‘seamless’ within this context
3. Formal proof of soundness of the approach of bootstrapping a trustworthy system from untrustworthy components
4. An open-source SysML v2 model of the proposed system architecture with prototype for demonstrating key functionality

→ Papers In-Progress

1. "Seamless Digital Engineering: Motivating a Grand Challenge"
   - *80% complete*
   - **Target conference:** INCOSE Western States Regional Conference 2023
2. "Digital Engineering Modeling Languages as LISP-Embedded Domain-Specific Languages"
   - *20% complete*
   - **Target conference:** INCOSE Western States Regional Conference 2023
3. "Architecture Essentials of a Seamless Digital Engineering System"
   - *50% complete*
   - **Target conference:** INCOSE IS 2024

Timeline to Completion

Summer 2023 JPL internship at half-time; continue HACK architecture modeling and prototype development; theory development, setup and testing of formal definitions and proofs; notification of acceptance for 2 conference papers

Fall 2023 Develop proofs, test and interpret results; architecture model completeness assessment; attend conference, and submit 1-2 papers to INCOSE IS conference or SE journal

Spring 2024 Final editing and presentation of results in dissertation; defense and graduation planned near end of semester

→ Future Work

- Continue architecture design and prototyping
- Reference model and reference architecture of DE lifecycle with product-line architecture instantiations for particular types of systems
- Computational type theory view and investigations of system architecture building from SysML v2 semantics
- Proving an architecture seamless and trustworthy at the type level
- Analysis of system bootstrap paths, size and effort
- Comparison and type-level analysis of extant computer system architectures
- Working out details of computational design of HACK systems
- Characterize ‘architectural ossification’ or intransigence

→ Thank you!

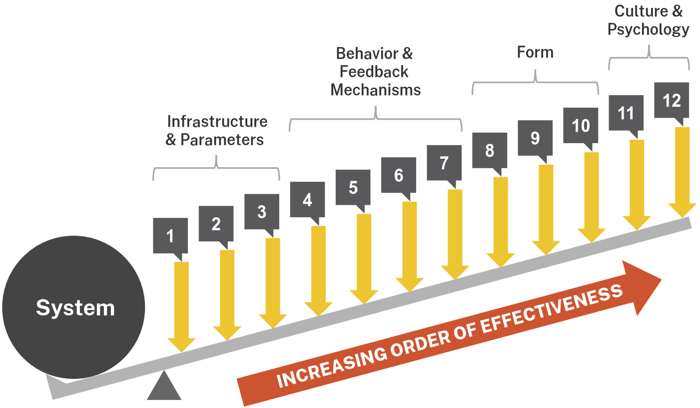


Figure: Places to intervene in a system (Adapted from Meadows 2008)

→ References

- S. Algarni (2021). "Cybersecurity Attacks: Analysis of "WannaCry" Attack and Proposing Methods for Reducing or Preventing Such Attacks in Future". *ICT Systems and Sustainability*. Ed. by M. Tuba, S. Akashe, and A. Joshi. Springer Singapore. ISBN: 978-981-15-8289-9. DOI: 10.1007/978-981-15-8289-9_73
- M. Ammar (2018). "ex uno pluria: The Service-Infrastructure Cycle, Ossification, and the Fragmentation of the Internet". *ACM SIGCOMM Computer Communication Review* 48.1. DOI: 10.1145/3211852.3211861
- A. W. Appel et al. (2017). "Position paper: the science of deep specification". *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 375.2104. DOI: 10.1098/rsta.2016.0331
- M. Bajaj, S. Friedenthal, and E. Seidewitz (2022). "Systems modeling language (SysML v2) support for digital engineering". *Insight* 25.1. DOI: 10.1002/inst.12367
- P. Borrello et al. (2022). "ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture". *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/borrello

→ References (continued)

- M. Bringolf, D. Winterer, and Z. Su (2022). "Finding and Understanding Incompleteness Bugs in SMT Solvers". *37th IEEE/ACM International Conference on Automated Software Engineering*. DOI: 10.1145/3551349.3560435
- S. Chiricescu et al. (2013). "SAFE: A clean-slate architecture for secure systems". *2013 IEEE International Conference on Technologies for Homeland Security (HST)*. DOI: 10.1109/THS.2013.6699066
- D. Chisnall (2018). "C Is Not a Low-level Language: Your computer is not a fast PDP-11.". *Queue* 16.2. ISSN: 15427730. DOI: 10.1145/3212477.3212479
- J. D. Claxton, C. Cavoli, and C. Johnson (2005). *Test and Evaluation Management Guide*. Tech. rep. Defence Acquisition University, Fort Belvoir, VA. URL: https://apps.dtic.mil/sti/pdfs/ADA436591.pdf
- D. Cofer (2021). "Cyber-assured systems engineering with AADL". *Retrieved June* 27. URL: https://apps.dtic.mil/sti/pdfs/AD1147940.pdf
- N. R. Council et al. (1999). *Trust in cyberspace*. National Academies Press. URL: http://people.eecs.berkeley.edu/~tygar/papers/Trust_in_Cyberspace.pdf

→ References (continued)

- X. Cui et al. (2022). "Toward Building and Optimizing Trustworthy Systems Using Untrusted Components: A Graph-Theoretic Perspective". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.5. ISSN: 1937-4151. DOI: 10.1109/TCAD.2021.3086765
- *META-II* (2010). Broad Agency Announcement. Defense Advanced Research Projects Agency
- *Clean-slate design of Resilient, Adapative, Secure Hosts (CRASH)* (2010). Broad Agency Announcement. Defense Advanced Research Projects Agency
- *Circuit Realization At Faster Timescales (CRAFT)* (2015). Broad Agency Announcement. Defense Advanced Research Projects Agency
- M. Dawson et al. (2021). "Understanding the Challenge of Cybersecurity in Critical Infrastructure Sectors". *Land Forces Academy Review* 26.1. DOI: 10.2478/raft-2021-0011. URL: https://doi.org/10.2478/raft-2021-0011

→ References (continued)

- O. Dedehayir and S. J. Mäkineif (2008). "Dynamics of Reverse Salience As Technological Performance Gap: an Empirical Study of the Personal Computertechnology System". *Journal of technology management & innovation* 3.3. DOI: 10.4067/S0718-27242008000100006
- C. Delp (2019). *Open Model-Based Engineering Environments*. Presentation. URL: https://www.nist.gov/system/files/documents/2019/04/05/14_delp.pdf
- J. X. K. L. Z. Du, Z. D. L. L. Q. Wu, and M. P. B. Mao (2023). "Silent Bugs Matter: A Study of Compiler-Introduced Security Bugs". *(preprint)*. URL: http://hexhive.epfl.ch/publications/files/23SEC4.pdf
- J. Easterly (2023). *CISA Director Easterly Remarks at Carnegie Mellon University*. URL: https://www.cisa.gov/cisa-director-easterly-remarks-carnegie-mellon-university
- M. Felleisen et al. (2018). "A programmable programming language". *Communications of the ACM* 61.3. DOI: 10.1145/3127323

→ References (continued)

- S. Friedenthal (2023). *SysML v2 Language Intro to Structure and Behavior Modeling*. SysML v2 Submission Team. URL: https://github.com/Systems-Modeling/SysML-v2-Release
- A. Hobbs (2021). "The colonial pipeline hack: Exposing vulnerabilities in us cybersecurity". *SAGE Business Cases*. SAGE Publications: SAGE Business Cases Originals. ISBN: 9781529789768. DOI: 10.4135/9781529789768
- T. P. Hughes (1993). *Networks of power: electrification in Western society, 1880-1930*. JHU press
- P. A. Karger and R. R. Schell (2002). "Multics security evaluation: Vulnerability analysis". *18th Annual Computer Security Applications Conference, 2002. Proceedings*. IEEE. ISBN: 0-7695-1828-1. DOI: 10.1109/CSAC.2002.1176286
- P. Kocher et al. (2020). "Spectre attacks: Exploiting speculative execution". *Communications of the ACM* 63.7
- M. Lipp et al. (2020). "Meltdown: Reading kernel memory from user space". *Communications of the ACM* 63.6

→ References (continued)

- A. M. Madni (2012). "Elegant systems design: Creative fusion of simplicity and power". *Systems Engineering* 15.3. DOI: 10.1002/sys.21209
- F. Massacci, T. Jaeger, and S. Peisert (2021). "Solarwinds and the challenges of patching: Can we ever stop dancing with the devil?" *IEEE security & privacy* 19.2. DOI: 10.1109/MSEC.2021.3050433
- J. McLean (1997). "Is the trusted computing base concept fundamentally flawed?" *IEEE Symposium on Security and Privacy*. IEEE Computer Society
- D. H. Meadows (2008). *Thinking in systems: A primer*. Chelsea Green Publishing. ISBN: 9781603580557. URL: https://books.google.com/books?id=CpbLAgAAQBAJ
- B. Mesmer et al. (2022). "Transdisciplinary Systems Engineering Approaches". *Recent Trends and Advances in Model Based Systems Engineering*. Ed. by A. M. Madni et al. Springer International Publishing. ISBN: 978-3-030-82083-1. DOI: 10.1007/978-3-030-82083-1_49

→ References (continued)

- J. S. Moore (2003). "A grand challenge proposal for formal methods: A verified stack". *Formal methods at the crossroads. From panacea to foundational support.* Ed. by B. Aichernig, T. Maibaum, and A. Haeberer. Lecture Notes in Computer Science. Springer. ISBN: 9783540205272. DOI: 10.1007/978-3-540-40007-3_11
- — (2007). *Piton: a mechanically verified assembly-level language*. Vol. 3. Springer. ISBN: 9780585336541
- C. Mundie et al. (2002). *Trustworthy computing*. Tech. rep. Microsoft
- R. Munroe (2020). *Dependency*. URL: https://xkcd.com/2347/
- K. Murdock et al. (2020). "Plundervolt: Software-based Fault Injection Attacks against Intel SGX". *2020 IEEE Symposium on Security and Privacy (SP)*. DOI: 10.1109/SP40000.2020.00057
- M. Watson, B. Mesmer, and P. Farrington (2020). *Engineering Elegant Systems: The Practice of Systems Engineering*. Tech. rep. National Aeronautics and Space Administration. URL: https://ntrs.nasa.gov/api/citations/20205003646/downloads/NASA_TP_20205003646_interactive.pdf

→ References (continued)

- G. Papastergiou et al. (2016). "De-ossifying the internet transport layer: A survey and future perspectives". *IEEE Communications Surveys & Tutorials* 19.1. DOI: 10.1109/COMST.2016.2626780
- O. S. P. Porras and R. Lindell (1995). "The Intel 80×86 Processor Architecture: Pitfalls for Secure Systems". *Proc. IEEE Symp. Security and Privacy*. ISBN: 9780818670152. DOI: 10.1109/SECPRI.1995.398934
- J. J. Rajendran, O. Sinanoglu, and R. Karri (2016). "Building Trustworthy Systems Using Untrusted Components: A High-Level Synthesis Approach". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.9. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2016.2530092
- J. A. Rees (1995). "A security kernel based on the lambda-calculus". PhD thesis. Massachusetts Institute of Technology. URL: http://hdl.handle.net/1721.1/5944
- J. E. Richardson, M. J. Carey, and D. T. Schuh (1993). "The design of the E programming language". *ACM Transactions on Programming Languages and Systems (TOPLAS)* 15.3. DOI: 10.1145/169683.174157

→ References (continued)

- S. van Schaik, A. Kwong, et al. (2020). *SGAxe: How SGX Fails in Practice*. https://sgaxeattack.com/
- S. van Schaik, M. Minkin, et al. (2021). "CacheOut: Leaking Data on Intel CPUs via Cache Evictions". *2021 IEEE Symposium on Security and Privacy (SP)*. DOI: 10.1109/SP40001.2021.00064
- M. Schwarz, S. Weiser, and D. Gruss (2019). "Practical Enclave Malware with Intel SGX". *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by R. Perdisci et al. Springer International Publishing. ISBN: 978-3-030-22038-9. DOI: 10.1007/978-3-030-22038-9_9
- M. Schwarz, S. Weiser, D. Gruss, et al. (2017). "Malware Guard Extension: Using SGX to Conceal Cache Attacks". *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by M. Polychronakis and M. Meier. Springer International Publishing. ISBN: 978-3-319-60876-1. DOI: 10.1007/978-3-319-60876-1_1
- D. Seal (2018). "The System Engineering 'V'–Is it Still Relevant in the Digital Age?" *Boeing Company, Global Product Data Interoperability Summit, Presentation*

→ References (continued)

- S. Sethumadhavan et al. (2015). "Trustworthy hardware from untrusted components". eng. *Communications of the ACM* 58.9. ISSN: 0001-0782. DOI: 10.1145/2699412
- D. Skarlatos et al. (2019). "MicroScope: Enabling Microarchitectural Replay Attacks". *Proceedings of the 46th International Symposium on Computer Architecture*. ISCA '19. Association for Computing Machinery. ISBN: 9781450366694. DOI: 10.1145/3307650.3322228. URL: https://doi.org/10.1145/3307650.3322228
- M. Smith and G. Mulrain (2018). "Equi-Failure: The National Security Implications of the Equifax Hack and a Critical Proposal for Reform". *Journal of National Security Law & Policy* 9. URL: https://ssrn.com/abstract=3253116
- E. Spafford (2004). *Exploring Grand Challenges in Trustworthy Computing*. URL: https://nitrd.gov/Subcommittee/lsn/material/20040316_lsn_spafford.pdf
- I. Thomas and B. A. Nejmeh (1992). "Definitions of tool integration for environments". *IEEE software* 9.2. DOI: 10.1109/52.120599
- K. Thompson (1984). "Reflections on trusting trust". *Communications of the ACM* 27.8. DOI: 10.1145/358198.358210

→ References (continued)

- J. Van Bulck et al. (2018). "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution". *Proceedings of the 27th USENIX Security Symposium*. See also technical report Foreshadow-NG Weisse et al. 2018. USENIX Association
- C. Voinea (2018). "Designing for conviviality". *Technology in Society* 52. Technology and the Good Society. ISSN: 0160-791X. DOI: 10.1016/j.techsoc.2017.07.002
- M. P. Ward (1994). "Language Oriented Programming". *Software – Concepts & Tools* 15. URL: http://www.gkc.org.uk/martin/papers/middle-out-t.pdf
- M. D. Watson (2017). "Engineering Elegant Systems: Design at the System Level". *Penn State University Graduate Seminar*. M17-6300
- M. D. Watson, M. Griffin, et al. (2014). "Building a path to elegant design". *Proceedings of the International Annual Conference of the American Society for Engineering Management*. American Society for Engineering Management (ASEM). ISBN: 9781634399890. URL: https://www.nasa.gov/sites/default/files/atoms/files/28_watson_building_a_path_to_elegant_design_0.pdf

→ References (continued)

M. D. Watson, B. Mesmer, and P. Farrington (2019). "Engineering Elegant Systems: Postulates, Principles, and Hypotheses of Systems Engineering". *Systems Engineering in Context*. Ed. by S. Adams et al. Springer International Publishing. ISBN: 978-3-030-00114-8. DOI: 10.1007/978-3-030-00114-8_40

O. Weisse et al. (2018). "Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution". *Technical report*. See also USENIX Security paper Foreshadow Van Bulck et al. 2018

D. A. Wheeler (2009). "Fully countering trusting trust through diverse double-compiling". PhD thesis. George Mason University

D. Winterer, C. Zhang, and Z. Su (2020). "Validating SMT solvers via semantic fusion". *Proceedings of the 41st ACM SIGPLAN Conference on programming language design and implementation*. DOI: 10.1145/3385412.3385985